Exercise 1:

Replace delete temp; with delete marker; in the remove() function.

```
// main.c
// This file contains the example program used in the
// gdb debugging tutorial.
#include <stdio.h>

int number_instantiated = 0;

struct Node {
public:
    Node(const int &value, Node *next = 0) : value_(value), next_(next) {
        printf("%s%d%s\n", "Creating Node, ",
                ++number_instantiated,
                " are in existence right now");
    }
    ~Node() {
        printf("%s%d%s\n", "Destroying Node, ",
                --number_instantiated,
                " are in existence right now");
        next_ = 0;
    }

    Node* next() const { return next_; }
    void next(Node *new_next) { next_ = new_next; };
    const int& value() const { return value_; }
    void value(const int &value) { value_ = value; }

private:
    Node();
    int value_;
    Node *next_;
};

struct LinkedList {
public:
    LinkedList () : head_(0) {};
    ~LinkedList () { delete_nodes (); };

    // returns 0 on success, -1 on failure
    int insert (const int &new_item) {
        return ((head_ = new Node(new_item, head_)) != 0) ? 0 : -1;
    }
```

```
    // returns 0 on success, -1 on failure
    int remove (const int &item_to_remove) {
        Node *marker = head_;
        Node *temp = 0; // temp points to one behind as we iterate

        while (marker != 0) {
            if (marker->value() == item_to_remove) {
                if (temp == 0) { // marker is the first element in the list
                    if (marker->next() == 0) {
                        head_ = 0;
                        delete marker; // marker is the only element in the list
                        marker = 0;
                    } else {
                    head_ = new Node(marker->value(), marker->next());
                    delete marker;
                    marker = 0;
                    }
                    return 0;
                } else {
                    temp->next (marker->next());
                    delete marker; //Here lies the issue for part 1
                    temp = 0;
                    return 0;
                }
            }
            temp = marker;
            marker = marker->next();
        }
    return -1; // failure
}

void print (void) {
    Node *marker = head_;
    while (marker != 0) {
        printf("%d\n", marker->value());
        marker = marker->next();
    }
}

private:
    void delete_nodes (void) {
        Node *marker = head_;
        while (marker != 0) {
            Node *temp = marker;
```

```
            delete marker;
            marker = temp->next();
        }
    }
    Node *head_;
};

int main (int argc, char **argv) {
    LinkedList *list = new LinkedList ();

    list->insert (1);
    list->insert (2);
    list->insert (3);
    list->insert (4);

    printf("%s\n", "The fully created list is:");
    list->print ();

    printf("\n%s\n", "Now removing elements:");
    list->remove (4);
    list->print ();
    printf("\n");

    list->remove (1);
    list->print ();
    printf("\n");

    list->remove (2);
    list->print ();
    printf("\n");

    list->remove (3);
    list->print ();

    delete list;

    return 0;
}
```

Exercise 2:

Replace delete marker; with delete temp->next(); in the remove() function to properly deallocate the memory.

```
// main.c
// This file contains the example program used in the
// gdb debugging tutorial.
#include <stdio.h>

int number_instantiated = 0;

struct Node {
public:
    Node(const int &value, Node *next = 0) : value_(value), next_(next) {
        printf("%s%d%s\n", "Creating Node, ",
                ++number_instantiated,
                " are in existence right now");
    }
    ~Node() {
        printf("%s%d%s\n", "Destroying Node, ",
                --number_instantiated,
                " are in existence right now");
        next_ = 0;
    }

    Node* next() const { return next_; }
    void next(Node *new_next) { next_ = new_next; };
    const int& value() const { return value_; }
    void value(const int &value) { value_ = value; }

private:
    Node();
    int value_;
    Node *next_;
};

struct LinkedList {
public:
    LinkedList () : head_(0) {};
    ~LinkedList () { delete_nodes (); };

    // returns 0 on success, -1 on failure
    int insert (const int &new_item) {
```

```
        return ((head_ = new Node(new_item, head_)) != 0) ? 0 : -1;
    }

    // returns 0 on success, -1 on failure
    int remove (const int &item_to_remove) {
        Node *marker = head_;
        Node *temp = 0; // temp points to one behind as we iterate

        while (marker != 0) {
            if (marker->value() == item_to_remove) {
                if (temp == 0) { // marker is the first element in the list
                    if (marker->next() == 0) {
                        head_ = 0;
                        delete marker; // marker is the only element in the list
                        marker = 0;
                    } else {
                    head_ = new Node(marker->value(), marker->next());
                    delete marker;
                    marker = 0;
                    }
                    return 0;
                } else {
                    temp->next (marker->next());
                    delete temp->next(); //Here lies the issue for part 2
                    temp = 0;
                    return 0;
                }
            }
            temp = marker;
            marker = marker->next();
        }
    return -1; // failure
}

void print (void) {
    Node *marker = head_;
    while (marker != 0) {
        printf("%d\n", marker->value());
        marker = marker->next();
    }
}

private:
    void delete_nodes (void) {
        Node *marker = head_;
```

```
        while (marker != 0) {
            Node *temp = marker;
            delete marker;
            marker = temp->next();
        }
    }
    Node *head_;
};

int main (int argc, char **argv) {
    LinkedList *list = new LinkedList ();

    list->insert (1);
    list->insert (2);
    list->insert (3);
    list->insert (4);

    printf("%s\n", "The fully created list is:");
    list->print ();

    printf("\n%s\n", "Now removing elements:");
    list->remove (4);
    list->print ();
    printf("\n");

    list->remove (1);
    list->print ();
    printf("\n");

    list->remove (2);
    list->print ();
    printf("\n");

    list->remove (3);
    list->print ();

    delete list;

    return 0;
}
```